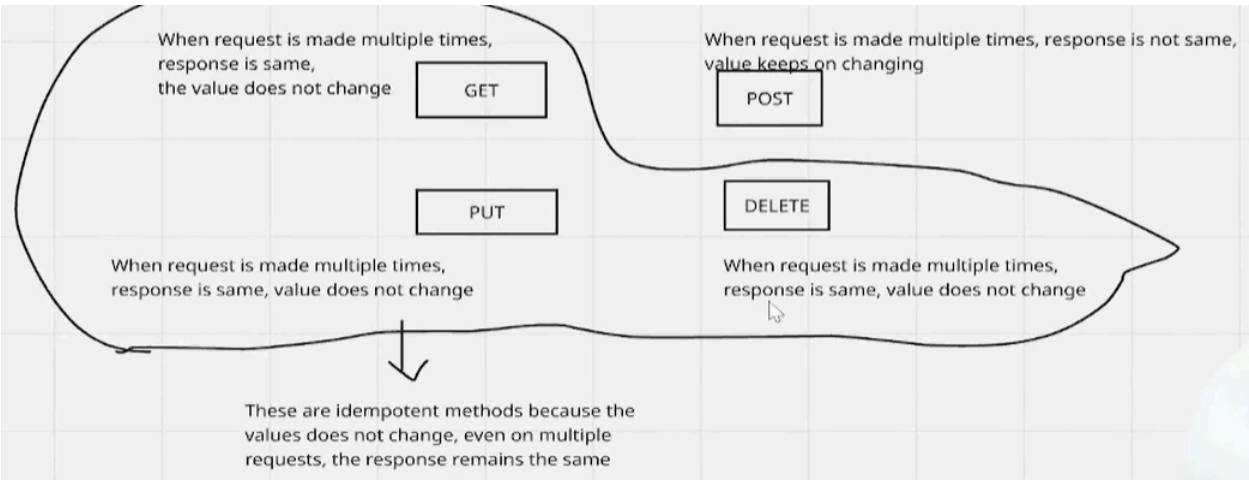
What is idempotency in API Testing?

In the context of **APIs** and **HTTP methods**, the term **idempotent** refers to an operation that can be performed multiple times without changing the result beyond the initial application.

In simpler terms:

> An idempotent HTTP method is one that has the same effect no matter how many times it is called.



🔍 Examples with HTTP Methods:

HTTP Method	Idempotent?	Explanation
GET	✅ Yes	Fetching the same resource multiple times doesn't change it.
PUT	✅ Yes	Updating a resource with the same data over and over has the same effect.
DELETE	✅ Yes	Deleting the same resource again and again doesn't change the state (once it's gone, it's gone).
POST	❌ No	Typically creates a new resource each time it's called — calling it multiple times creates duplicates.

HTTP Methods

GET – "Give me something"

- **What it does:** Asks the server to **send back data.**
- **Example:** Loading a webpage, viewing a user profile, checking your messages.
- **Analogy:** Looking at a menu at a restaurant.

🟡 POST – "Create something new"

- **What it does:** Sends data to the server to **create** something new.
- **Example:** Signing up for an account, making a new post on social media, submitting a contact form.
- **Analogy:** Telling a waiter your order so they can **make it for you.**

🔵 PUT – "Replace or update completely"

- **What it does:** Sends data to the server to **replace** something that already exists.
- **Example:** Updating your full profile (name, email, photo, etc.).
- **Analogy:** Replacing an old resume with a brand new one.

🟠 PATCH – "Update a part of something"

- **What it does:** Sends data to the server to **update just part** of something.
- **Example:** Changing just your profile picture or updating just your email.
- **Analogy:** Crossing out your old phone number on your resume and writing a new one.

🔴 DELETE – "Remove something"

- **What it does:** Tells the server to **delete** a resource.
- **Example:** Deleting a tweet, removing an item from your cart, closing your account.
- **Analogy:** Telling the waiter, "Cancel that order."

🟦 PUT vs PATCH — Main Difference

Feature	PUT	PATCH
Purpose	Replaces the **entire resource**	Updates **only part of the resource**
Idempotent	✅ Yes (safe to repeat)	✅ Usually yes (depends how it's used)
Data needed	Full updated object	Only the fields you want to change
Overwrites?	Yes, it overwrites the entire object	No, it just changes specific fields

📄 Example Scenario:

Let's say you have a user profile like this:

JSON

```
json
CopyEdit
{
  "username": "john_doe",
  "email": "john@example.com",
  "bio": "Loves hiking"
}
```

✅ Using PUT (Full Update)

If you just want to change the **bio**, with `PUT` you'd need to send the **whole object again**, even the parts that didn't change:

```
json
CopyEdit
PUT /user/123

{
  "username": "john_doe",
  "email": "john@example.com",
  "bio": "Coffee addict ☕"
}
```

> ✅ Replaces the entire profile with the new version.

✅ Using PATCH (Partial Update)

If you only want to change the **bio**, with `PATCH`, you just send what's changed:

```
json
CopyEdit
PATCH /user/123

{
  "bio": "Coffee addict ☕"
}
```

> ✅ Only the bio field is updated — the rest stays untouched.

🧠 When to Use Which?

- Use **PUT** when you're replacing the **whole resource**.
- Use **PATCH** when you're doing a **partial update**.

What is an Environment in API Testing?

An **environment** is like a **setup or workspace** where you test your API.

It includes important things like:

- The **base URL** (where your API is hosted)
- Any **variables** (like API keys, user tokens, etc.)
- Special **settings or data** the API might need

🔄 Why do we use environments?

When you're testing an API, it may be running in different places for different purposes:

Environment	What it's for
👨‍💻 **Development**	Where developers build and test new features
🧪 **Testing/QA**	Where testers (like you!) test the API safely
🚀 **Staging**	Like a "dress rehearsal" — final test before going live
🌍 **Production**	The real thing — where actual users use the API

📌 Real-life Example:

Imagine you're testing a **login API**:

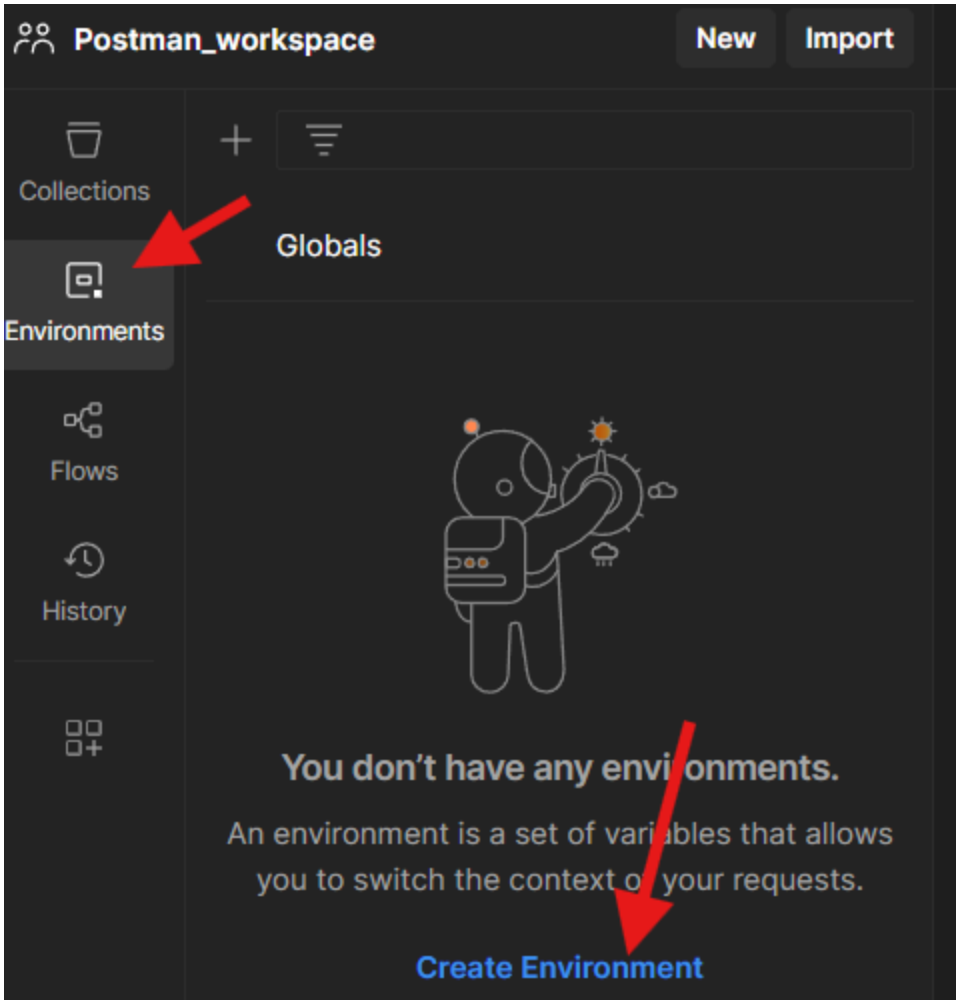
```
http
CopyEdit
POST /login
```

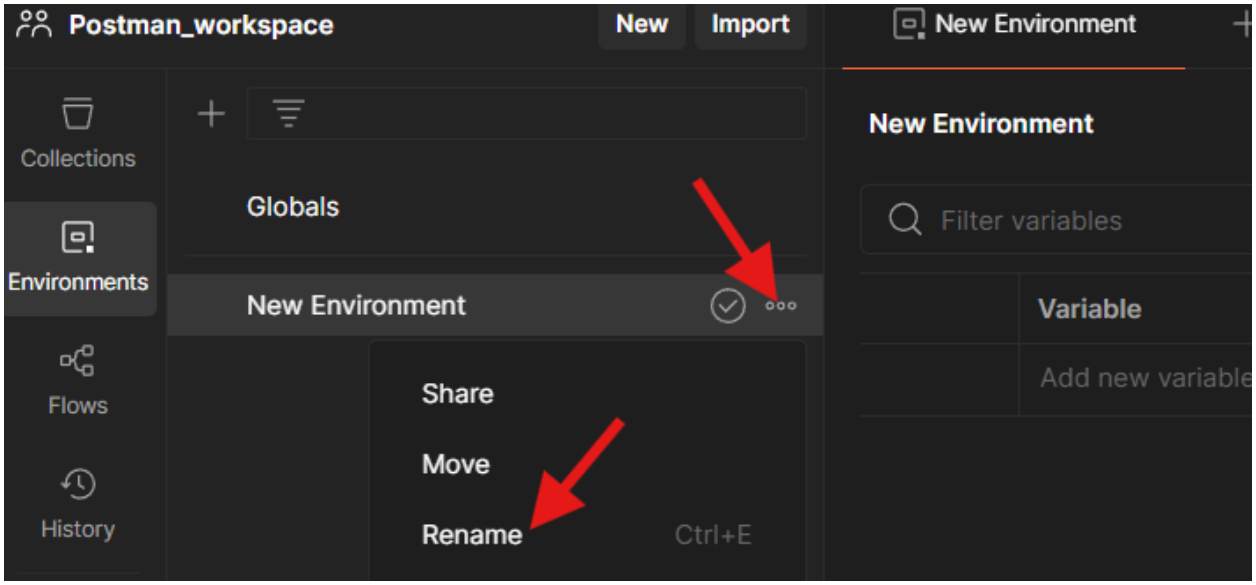
But this API exists in **different environments**:

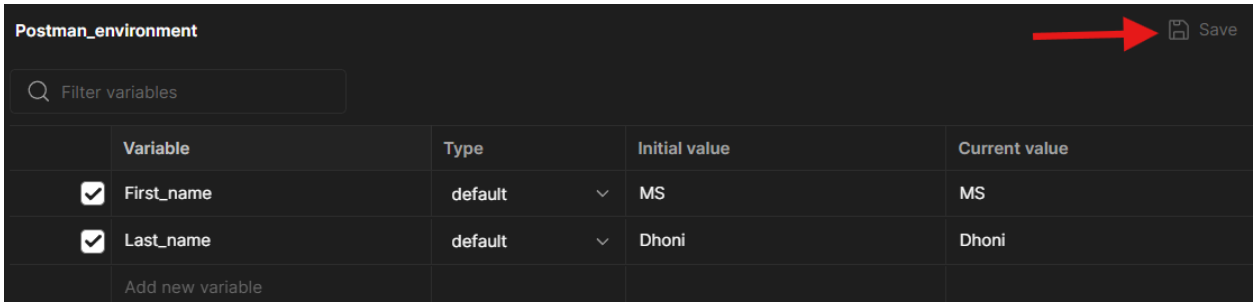
Environment	URL
Dev	`https://dev.api.myapp.com/login`
QA/Test	`https://test.api.myapp.com/login`
Prod	`https://api.myapp.com/login`

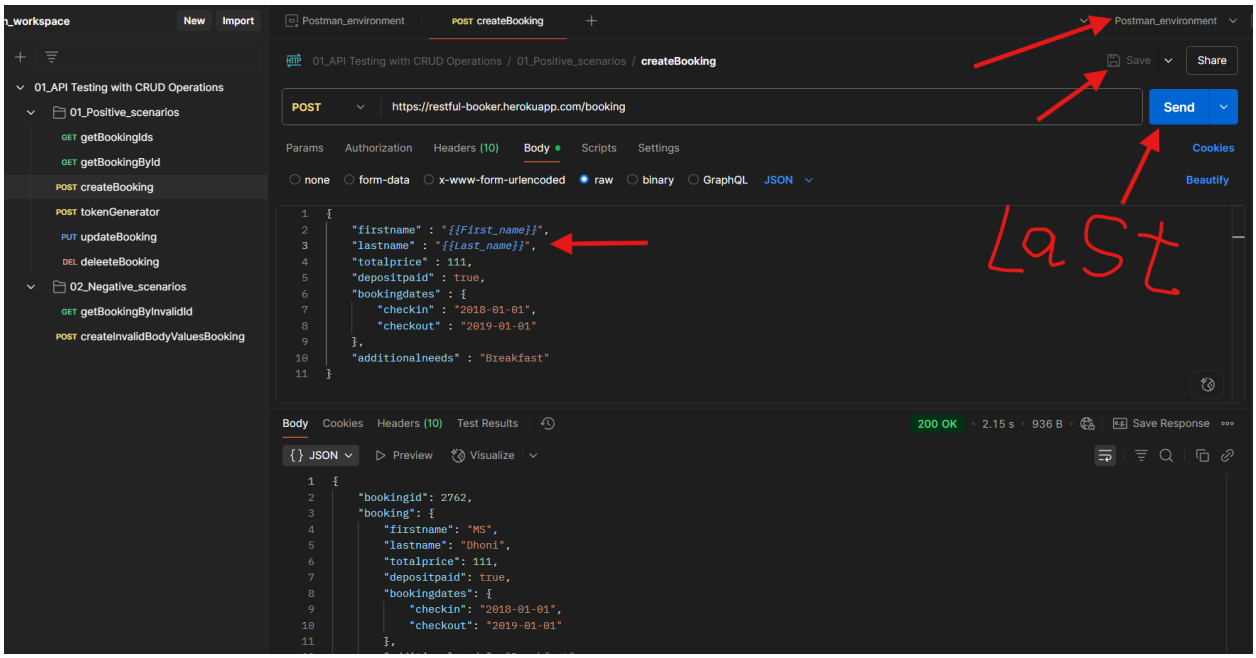
Create Environment

We are copying the id and tokens to delete request. To avoid copying we can use environment.

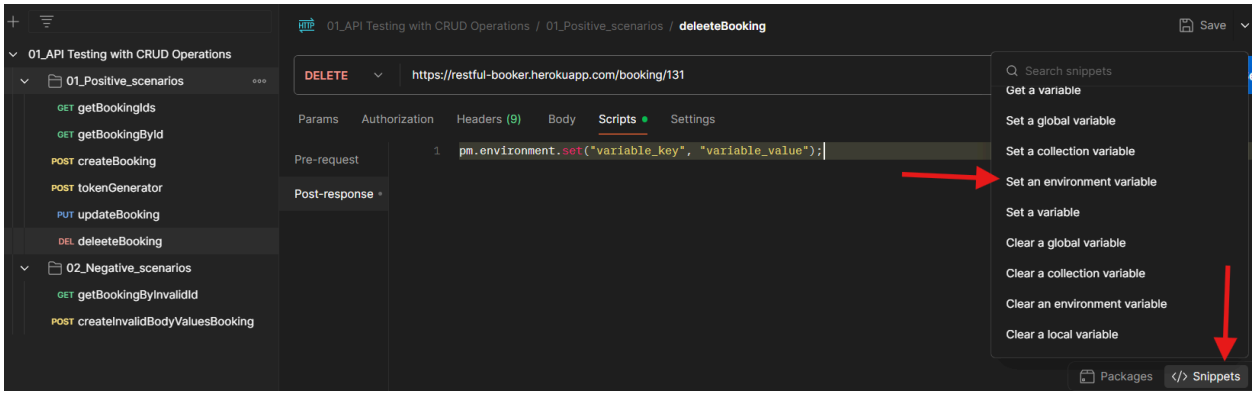

Postman_workspace
New
Import
Collections
Environments
Flows
History
Globals
You don’t have any environments.
An environment is a set of variables that allows you to switch the context of your requests.
Create Environment


Postman_workspace
New
Import
New Environment
Collections
Environments
Flows
History
Globals
New Environment
Share
Move
Rename
Ctrl+E
New Environment
Filter variables
Variable
Add new variable

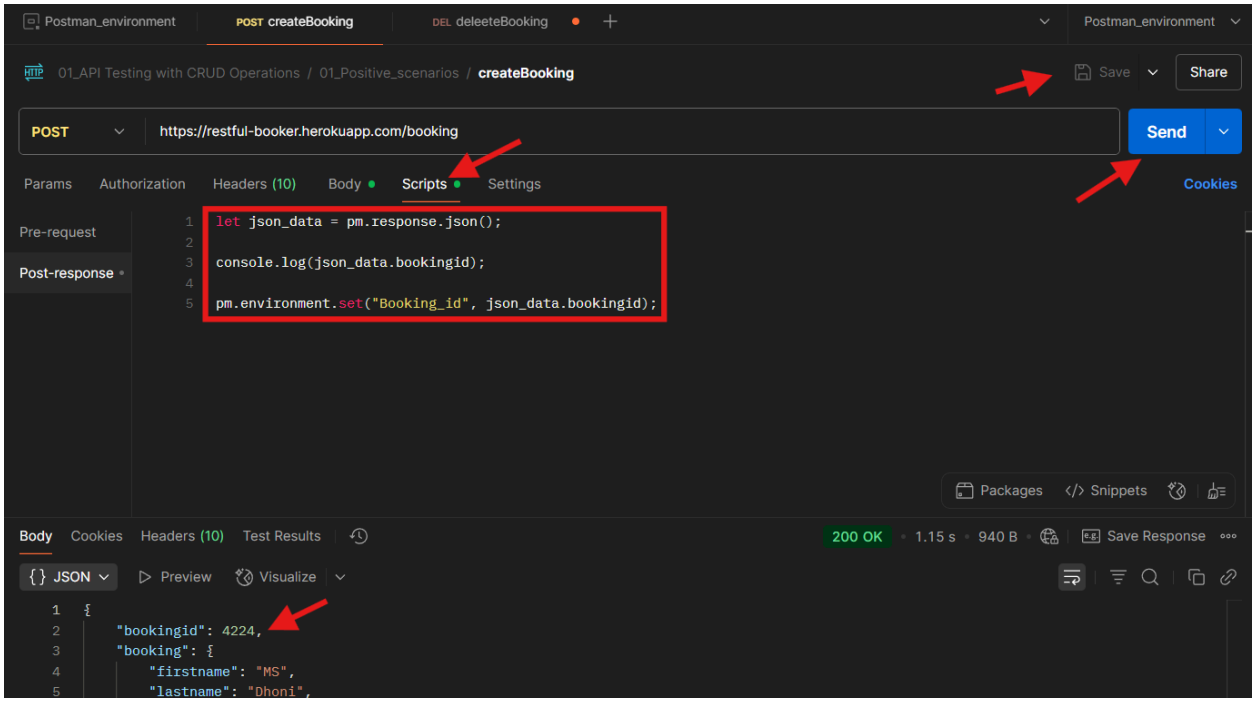




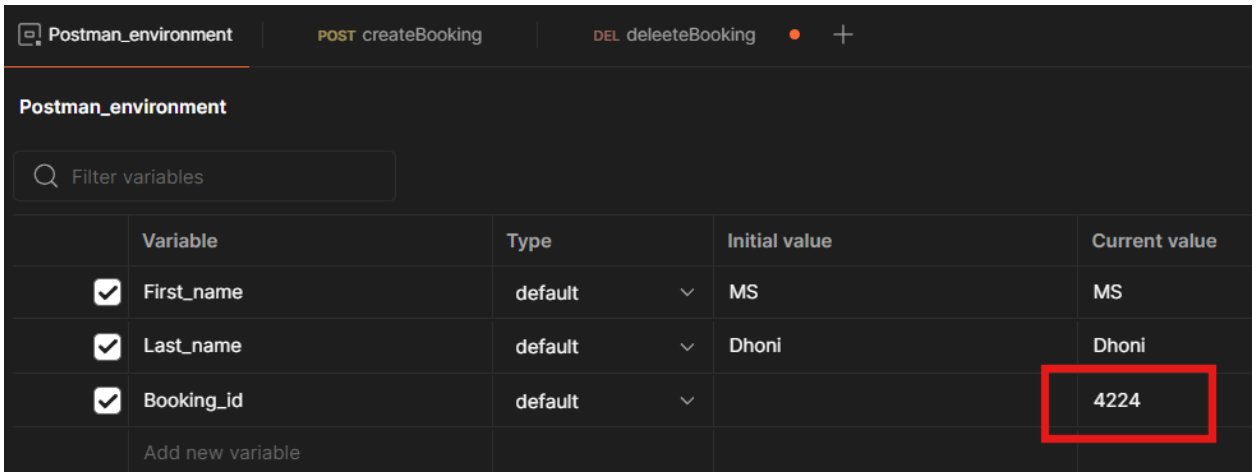
To set an Environment variable choose the scripts, snippets.



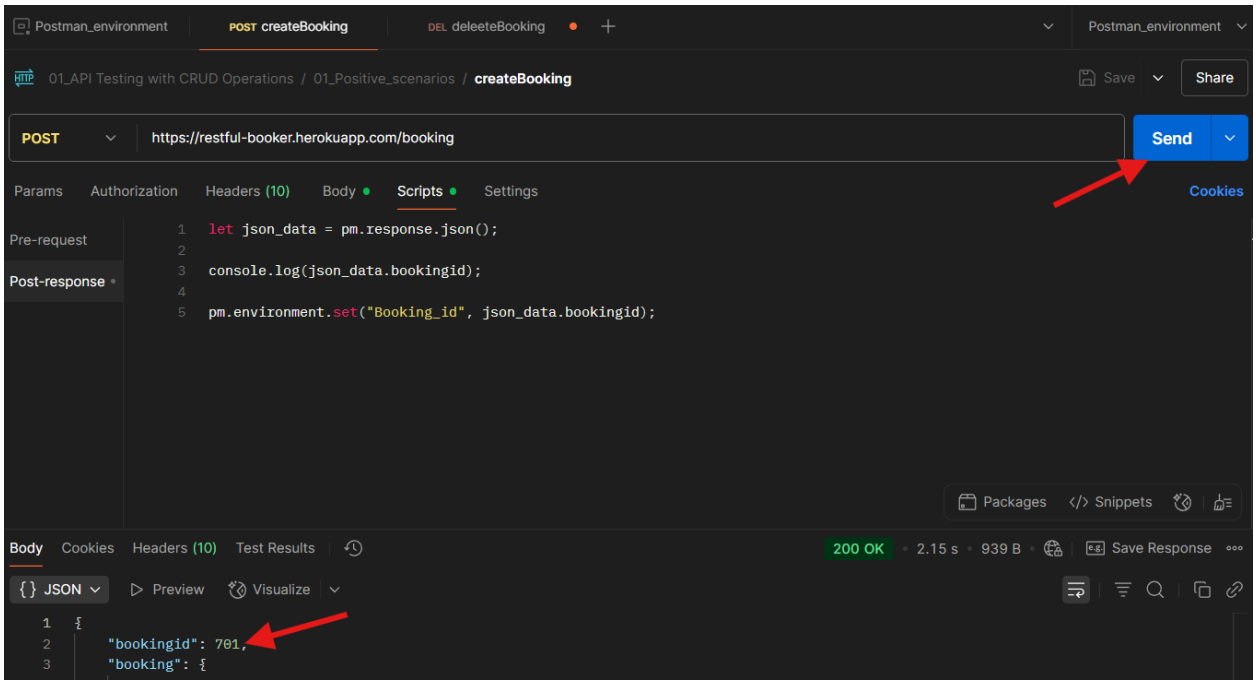
In the Post Request that makes the response

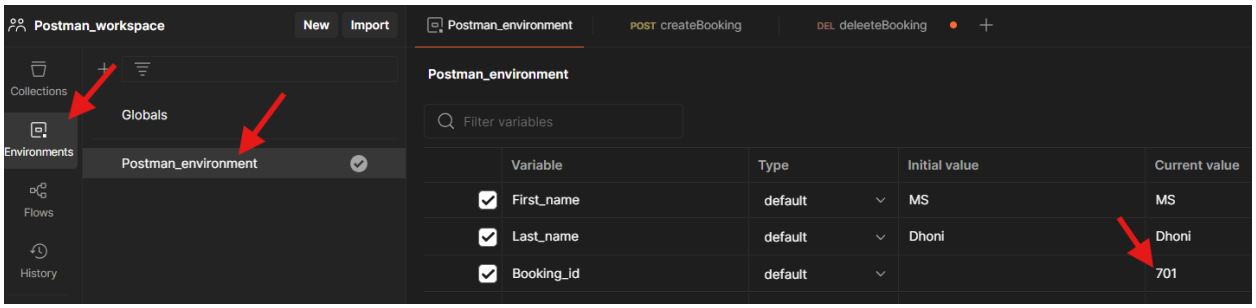


Adds to the environment:

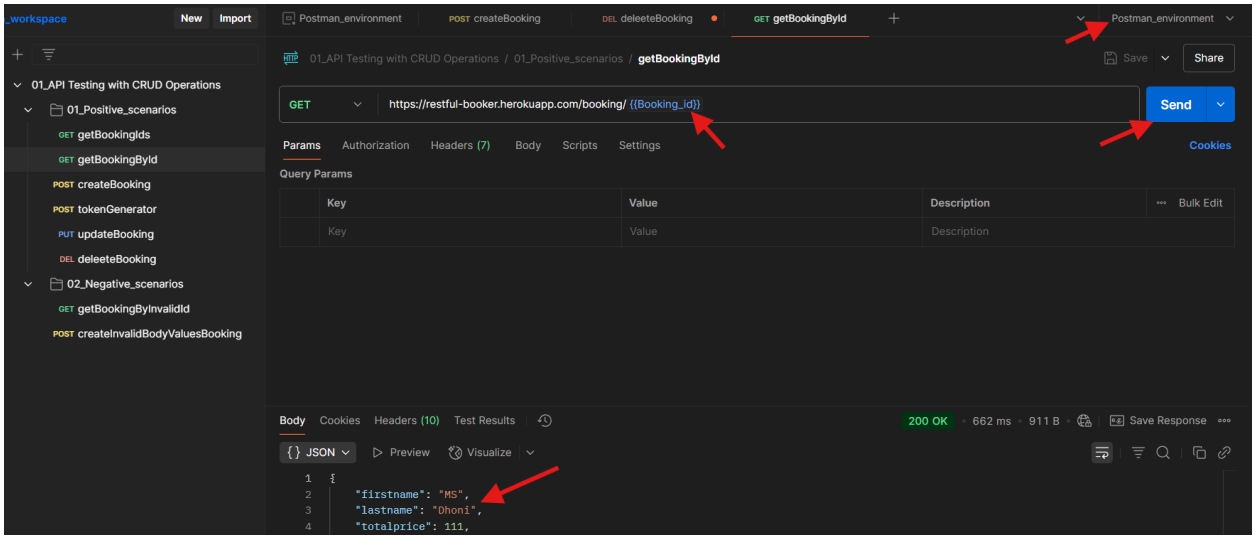


Every time id changes:

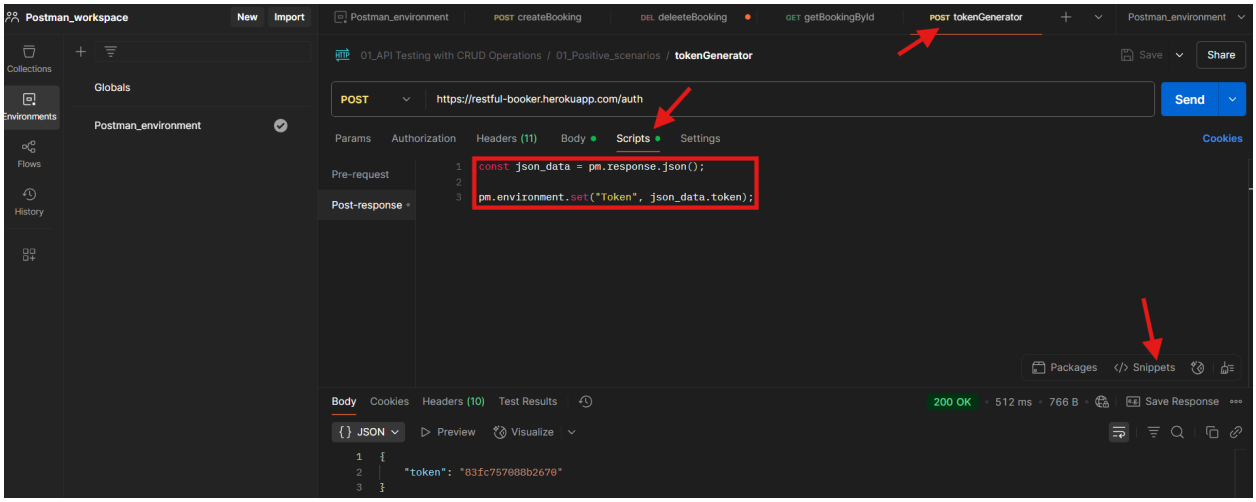


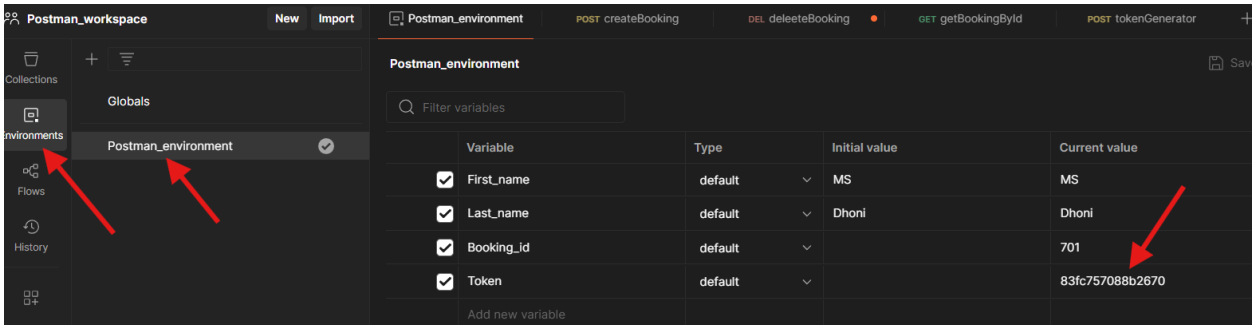


Now we can change the id of Get Booking by Id:



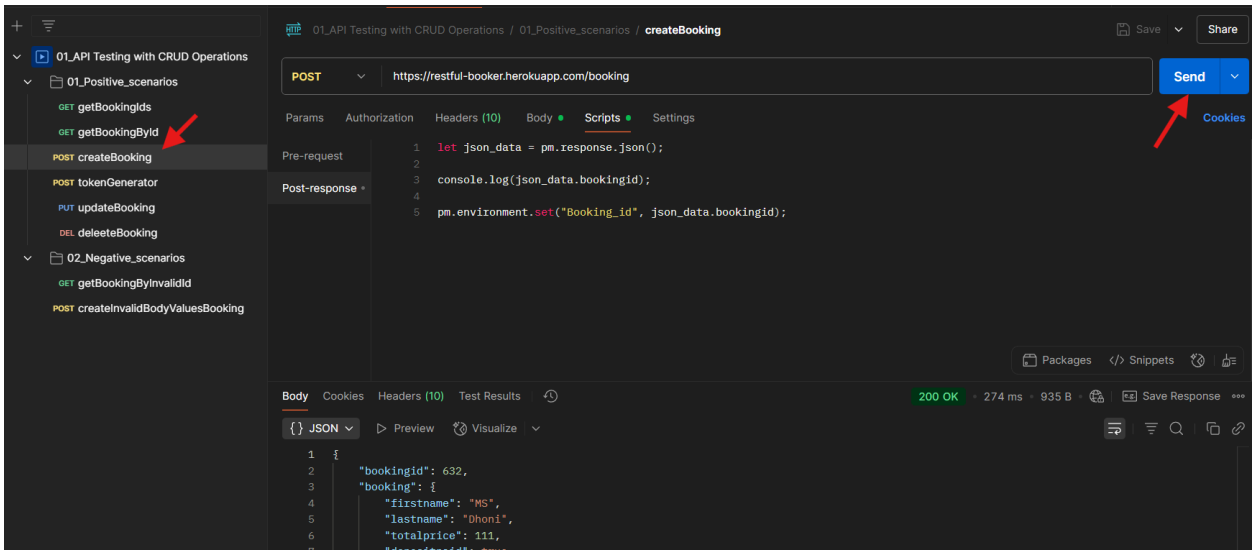
To add the token to the environment variable:

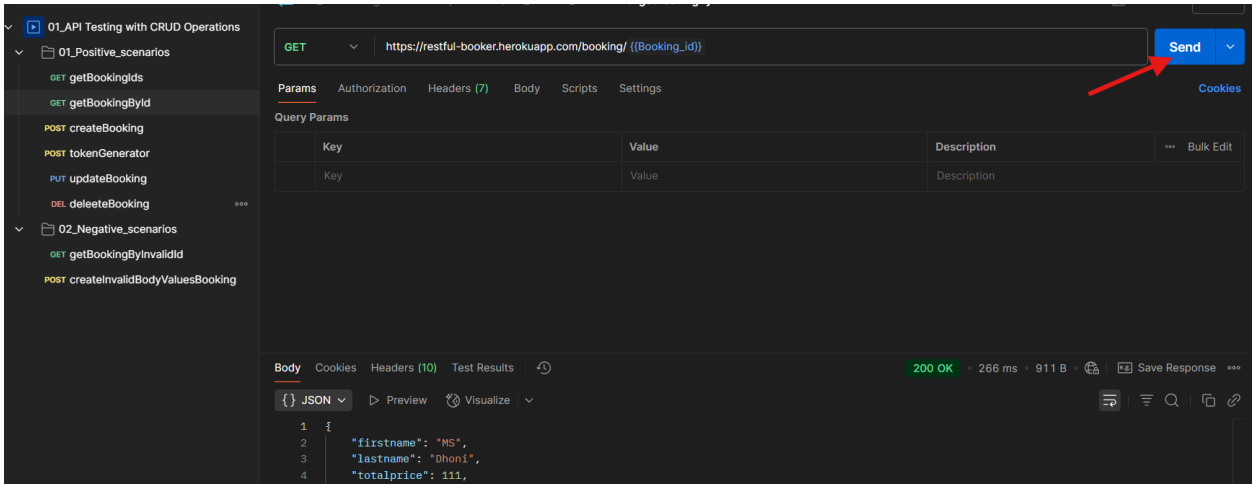


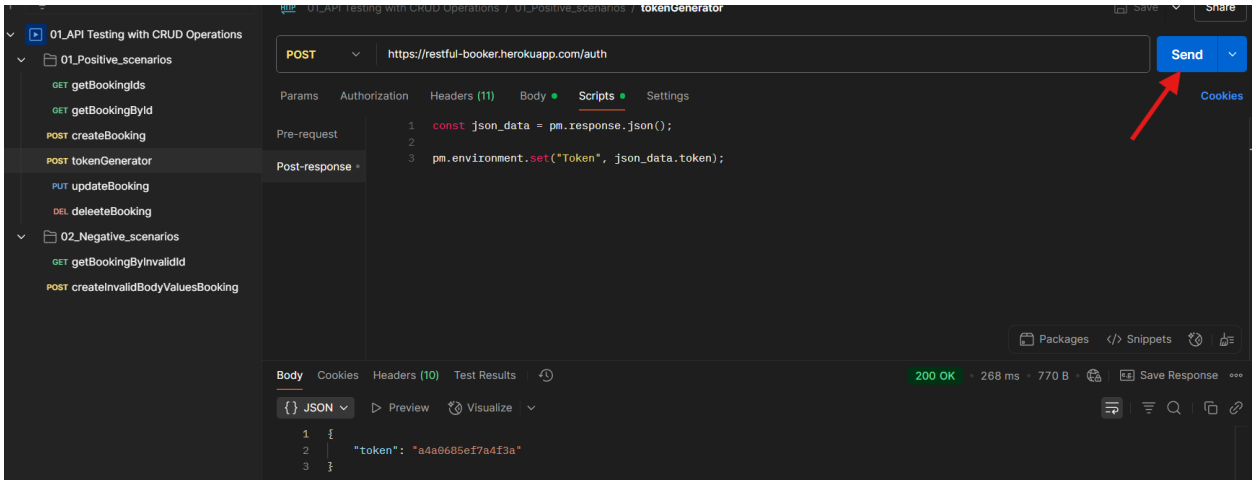


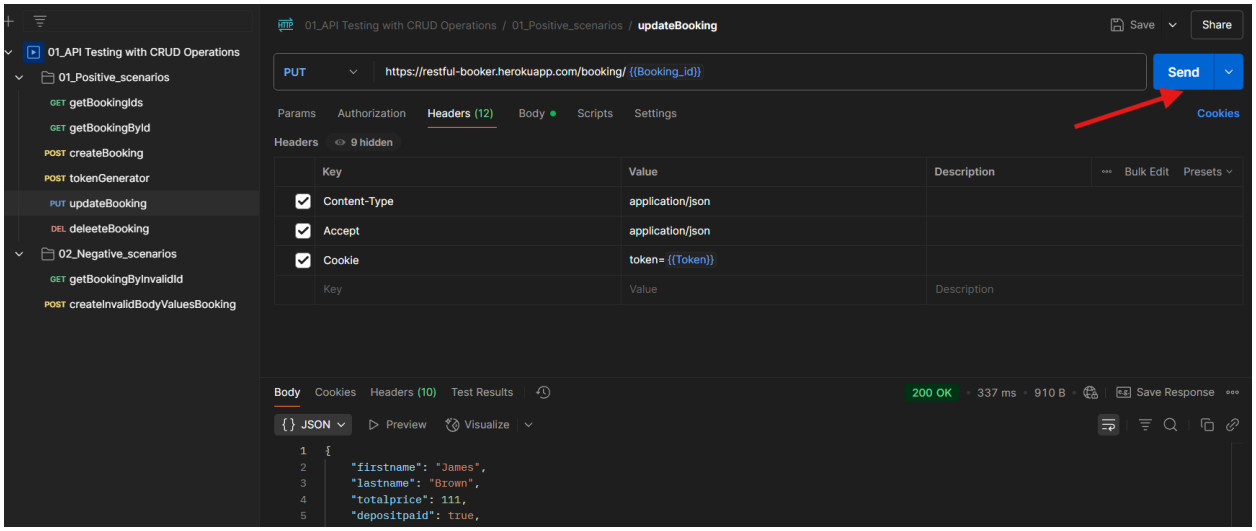
Token actually keeps on changing. Fixed response for idempotent.

Now You don't have to copy and paste the token and id. Example given below:

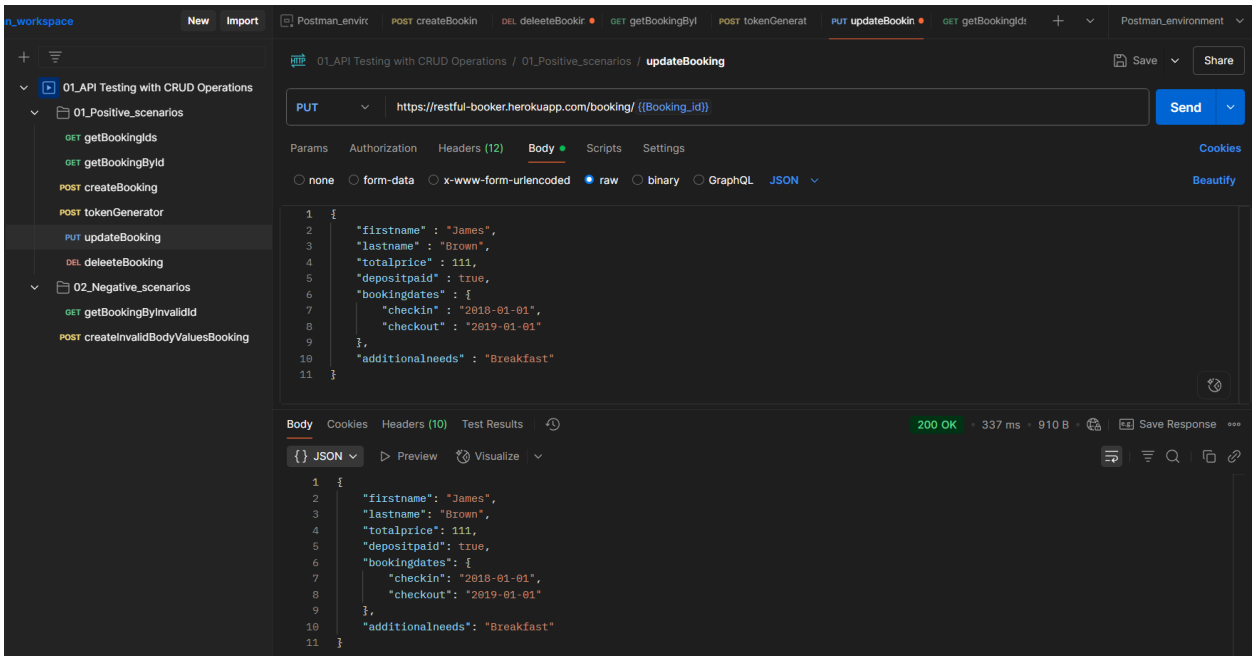




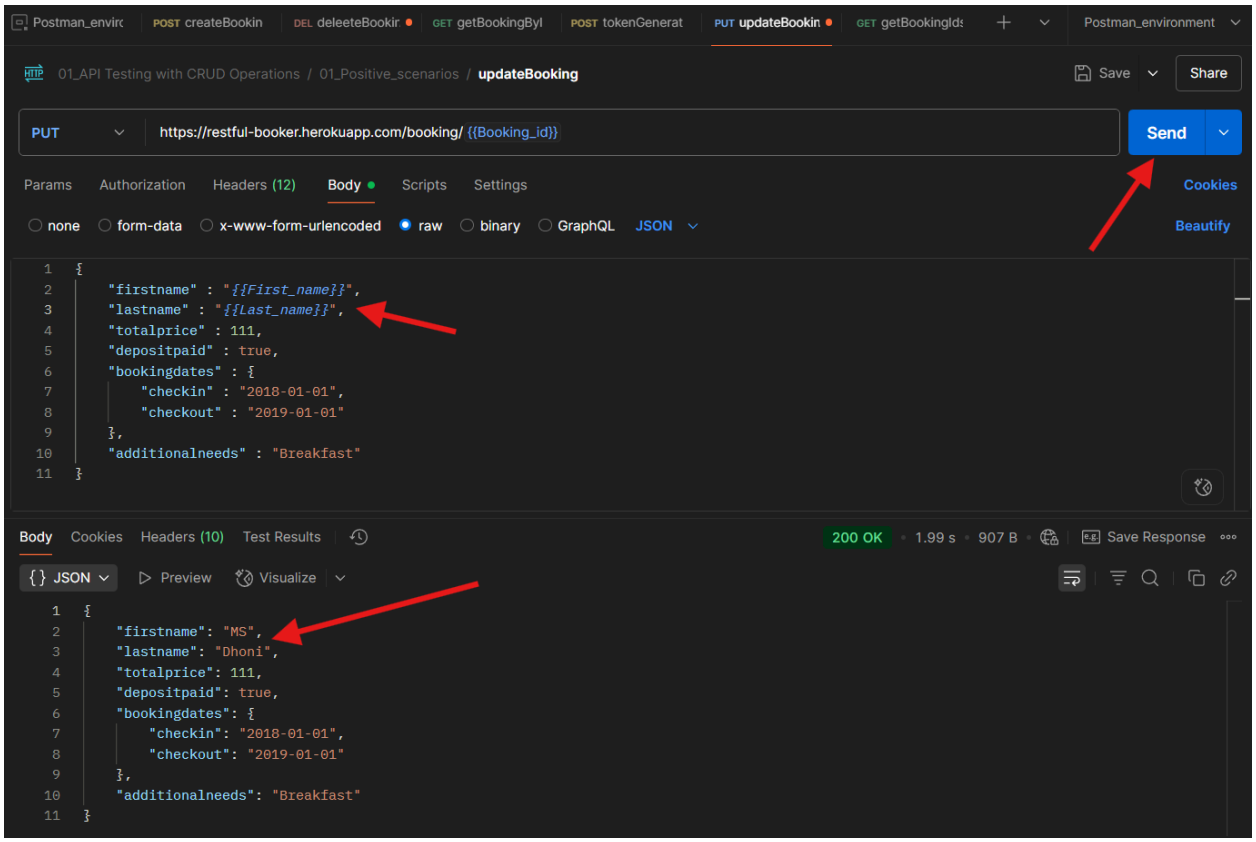


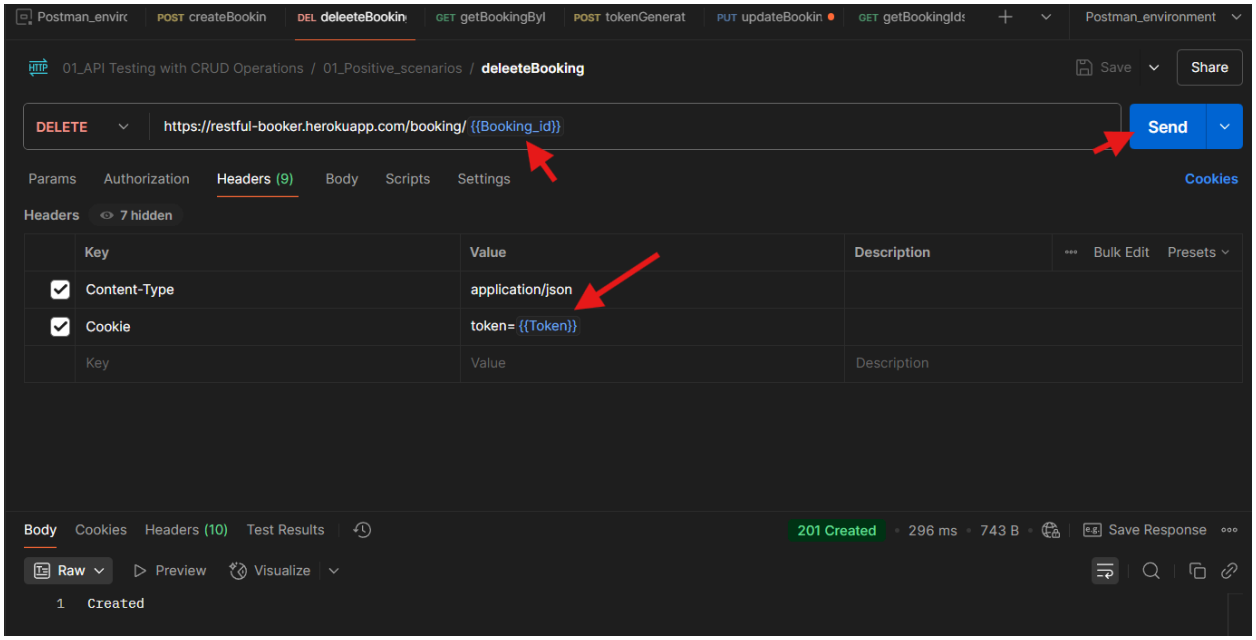


Its showing james brown because



After Updating the firstname and lastname





If you're using a tool like **Postman**, you can create **environments** and just change the **base URL variable**, like this:

```
js
CopyEdit
{{base_url}}/login
```

Then you define `base_url` for each environment, and switch easily. ✅

🎯 Why this is helpful:

- You **don't have to rewrite** your tests every time.
- You can safely test without affecting **real users**.
- You can **switch environments quickly** and test in different situations.

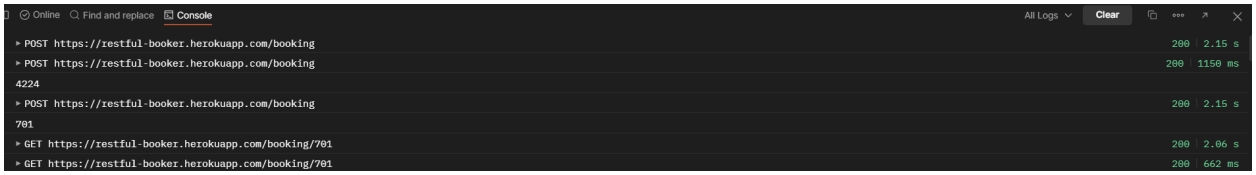
Console

What is the **Console** in Postman?

The **console** in Postman is like a small window where you can **see what's really happening** when you send an API request.

✅ Why do we use it?

1. **See details of your request**
   - What you sent
   - What you got back
   - Any errors
2. **Check if variables are working**
   - You can print variables like tokens or URLs to make sure they're correct.
3. **Find problems**
   - If something is not working, the console shows what went wrong (like wrong URL, missing data, etc.)
4. **Debug your test scripts**
   - You can write `console.log("Hello")` in your script and see it in the console — just like checking if something is working.



CTRL + ALT + C to open the console.

📌 Simple Example:

Let's say your test script has this line:

```
js
CopyEdit
console.log("Token is:", pm.environment.get("token"));
```

When you run the request, the console will show:

```
csharp
CopyEdit
Token is: abc123xyz
```

1. Which of the following HTTP methods is *not* idempotent?

A. GET

B. PUT

C. DELETE

D. POST

✅ **Correct Answer:** D. POST

🟦 **Explanation:** POST typically creates a new resource each time it's called, which changes the state, so it's not idempotent.

2. What does it mean if an HTTP method is *idempotent*?

A. It never returns an error.

B. It always returns the same response format.

C. It has the same effect no matter how many times it's called.

D. It only works on DELETE operations.

✅ **Correct Answer:** C. It has the same effect no matter how many times it's called.

🟦 **Explanation:** Idempotent methods can be safely repeated without changing the server's state after the initial call.

3. Which HTTP method would you use to partially update a user's profile?

A. GET

B. PUT

C. POST

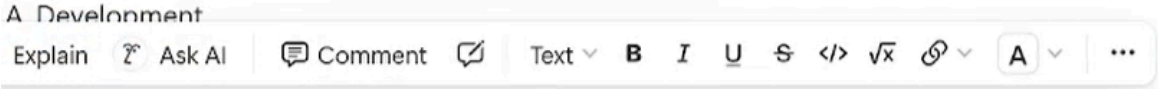
D. PATCH

✅ **Correct Answer:** D. PATCH

🟦 **Explanation:** PATCH is used to update part of a resource — like just the user's email or bio.

4. Which environment is typically used as a "final test" before going live in API testing?

A. Development



C. Staging

D. QA/Test

✅ **Correct Answer:** C. Staging

🟦 **Explanation:** Staging is like a dress rehearsal — used for final testing in a near-live setup.

Assertions in API Testing

📌 What is an Assertion?

An **assertion** is a **statement that validates** whether a specific condition is true or false during testing.

In **API testing**, assertions help confirm that the response from an API meets expected criteria.

🧪 Why Are Assertions Important?

- **Verify correctness** of API responses
- Catch **bugs and issues** early
- Validate **data integrity** and **response format**
- Ensure **performance** and **reliability**

🔍 Common Types of Assertions in API Testing

Assertion Type	What It Checks	Example
✅ Status Code	Is the HTTP status code correct?	`200 OK`, `404 Not Found`
📝 Response Body	Does the body contain expected data or text?	Includes `"success"` or `user_id`
📄 Headers	Are required headers present and correct?	`Content-Type`, `Authorization`
⏱️ Response Time	Is the API responding within acceptable time limits?	`< 1000ms`
🔐 Authentication	Is access restricted/protected as expected?	`401 Unauthorized`

🛠️ How Assertions Work in Postman

In Postman, you write assertions using the `pm` object (Postman's scripting interface).

Example:

```
javascript
CopyEdit
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

This checks that the response has a **200 status code**.

🧩 Assertion Libraries in Use

Postman uses **ChaiJS** (a JavaScript assertion library), which allows:

- **BDD-style** assertions like `to.have`, `to.be.below`, `to.include`, etc.

🧠 Best Practices for Using Assertions

- Use **clear test names** (describe what you're testing)
- Check **status code**, **response body**, **headers**, and **performance**
- Cover **positive and negative** scenarios

Postman Test Script Notes

✅ 1. Status Code Validation

```
javascript
CopyEdit
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

- **Purpose**: Verifies the response has a **200 OK** status.
- **Use Case**: Ensures the API call was **successful**.

🔍 2. Response Body Content Check

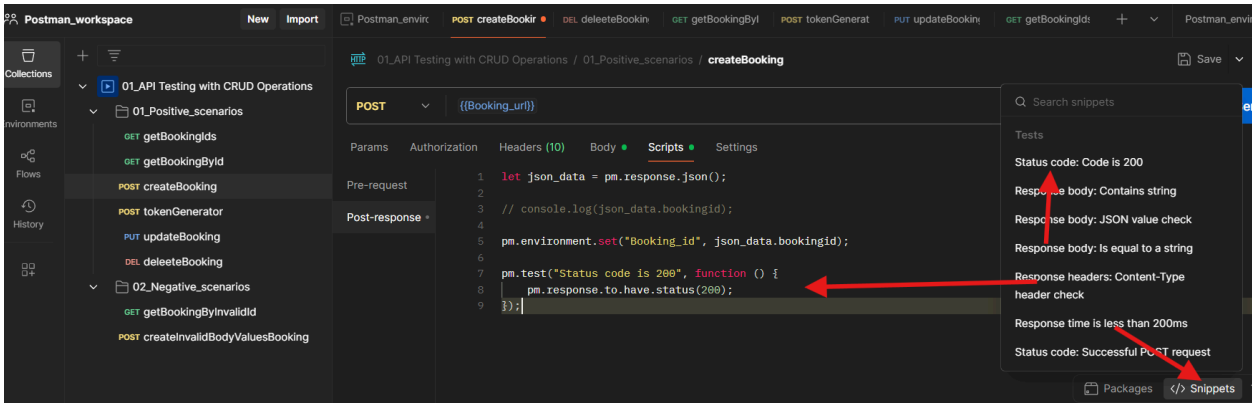
```
javascript
CopyEdit
pm.test("Body matches string", function () {
    pm.expect(pm.response.text()).to.include("Dhoni");
});
```

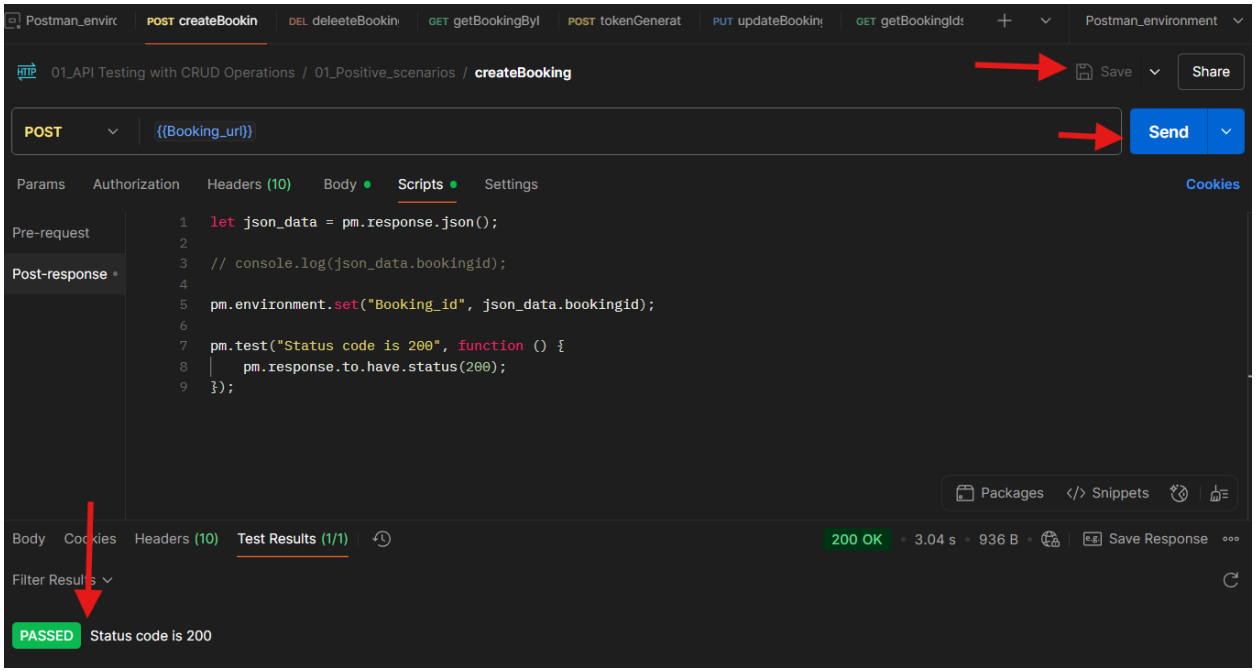
- **Purpose**: Checks if the **response body** contains the text `"Dhoni"`.
- **Use Case**: Confirms specific **data/content** is returned by the API.

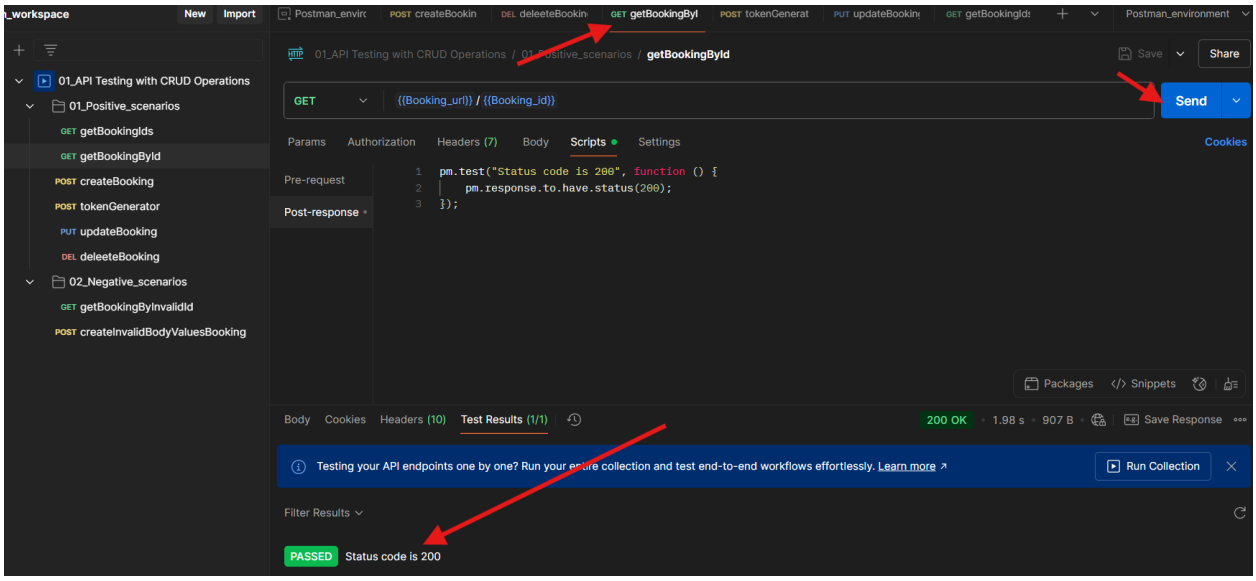
📄 3. Header Validation

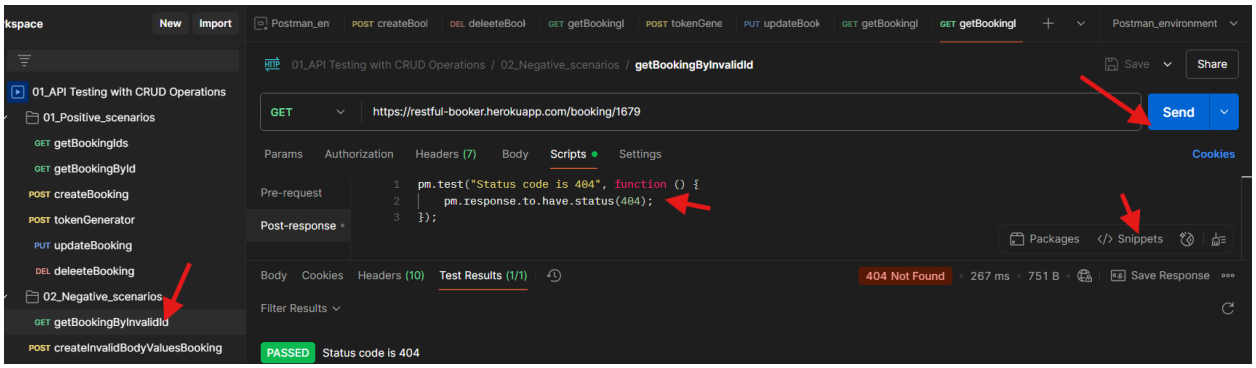
```
javascript
CopyEdit
pm.test("Content-Type is present", function () {
    pm.response.to.have.header("Content-Type", "application/json; charset=utf-8");
});
```

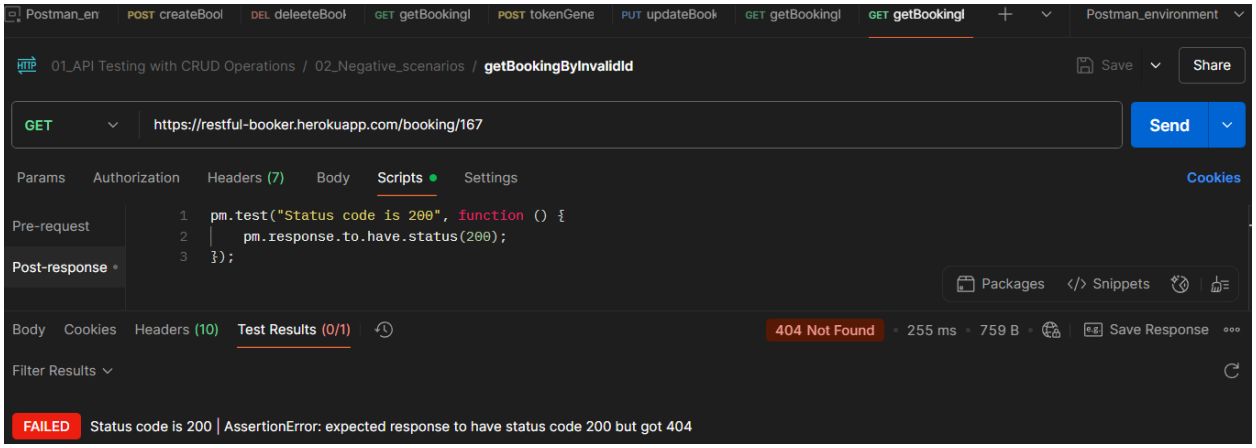
- **Purpose**: Verifies the **Content-Type header** is correct.
- **Use Case**: Ensures the response is in **JSON format**.




Postman_environment
01_API Testing with CRUD Operations / 01_Positive_scenarios / createBooking
Save
Share
POST
{{Booking_url}}
Send
Params
Authorization
Headers (10)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
let json_data = pm.response.json();
// console.log(json_data.bookingid);
pm.environment.set("Booking_id", json_data.bookingid);
pm.test("Status code is 200", function () {
pm.response.to.have.status(200);
});
Packages
Snippets
Body
Cookies
Headers (10)
Test Results (1/1)
200 OK
3.04 s
936 B
Save Response
Filter Results
PASSED
Status code is 200


New
Import
01_API Testing with CRUD Operations
01_Positive_scenarios
GET getBookingIds
GET getBookingById
POST createBooking
POST tokenGenerator
PUT updateBooking
DEL deleeteBooking
02_Negative_scenarios
GET getBookingByInvalidId
POST createInvalidBodyValuesBooking
01_API Testing with CRUD Operations / 01_Positive_scenarios / getBookingById
GET
{{Booking_url}} / {{Booking_id}}
Send
Share
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
pm.test("Status code is 200", function () {
pm.response.to.have.status(200);
});
Packages
Snippets
Body
Cookies
Headers (10)
Test Results (1/1)
200 OK
1.98 s
907 B
Save Response
Testing your API endpoints one by one? Run your entire collection and test end-to-end workflows effortlessly. Learn more
Run Collection
Filter Results
PASSED
Status code is 200


New
Import
01_API Testing with CRUD Operations
01_Positive_scenarios
GET getBookingIds
GET getBookingById
POST createBooking
POST tokenGenerator
PUT updateBooking
DEL deleeteBooking
02_Negative_scenarios
GET getBookingByInvalidId
POST createInvalidBodyValuesBooking
01_API Testing with CRUD Operations / 02_Negative_scenarios / getBookingByInvalidId
GET
https://restful-booker.herokuapp.com/booking/1679
Send
Share
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
pm.test("Status code is 404", function () {
pm.response.to.have.status(404);
});
Packages
Snippets
Body
Cookies
Headers (10)
Test Results (1/1)
404 Not Found
267 ms
751 B
Save Response
Filter Results
PASSED
Status code is 404



1. What is the primary purpose of an assertion in API testing?

A. To create API endpoints

B. To encrypt the API request

C. To validate that a condition is true or false

D. To generate fake API responses

> ✅ Correct Answer: C — Assertions validate whether a condition is true or false.

2. Which of the following assertions would be used to check API response speed?

A. Response Body

B. Status Code

C. Response Time

D. Headers

> ✅ Correct Answer: C — Response Time assertions ensure the API responds within acceptable limits.

3. In Postman, which JavaScript object is commonly used to write test assertions?

A. `assert`

B. `pm`

C. `chai`

D. `http`

> ✅ Correct Answer: B — Postman uses the pm object to write assertions.

4. What does the following Postman test script validate?

```
javascript
CopyEdit
pm.test("Body matches string", function () {
    pm.expect(pm.response.text()).to.include("Dhoni");
});
```

A. The response is in JSON format

B. The response status is 200

C. The response body contains the text "Dhoni"

D. The API took less than 1000ms to respond

> ✅ Correct Answer: C — This assertion checks for the presence of "Dhoni" in the response body.